

EL685271524

PTO/SB/17 (08-00)

Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL
for FY 2000

Patent fees are subject to annual revision.

Complete if Known

Application Number	
Filing Date	
First Named Inventor	Yadav
Examiner Name	
Group Art Unit	
Attorney Docket No.	MSI-6015US

TOTAL AMOUNT OF PAYMENT (\$)
1162.00**METHOD OF PAYMENT (check one)**

- 1.
- ☒
- The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number	12-0769
Deposit Account Name	Lee & Hayes PLLC

☒ Charge Any Additional Fee Required Under 37 CFR 1.16 and 1.17☐ Applicant claims small entity status. See 37 CFR 1.27

- 2.
- ☒
- Payment Enclosed:

☒ Check ☐ Credit card ☐ Money Order ☐ Other**FEE CALCULATION****1. BASIC FILING FEE**

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
101 690	201 345	Utility filing fee	690
106 310	206 155	Design filing fee	
107 480	207 240	Plant filing fee	
108 690	208 345	Reissue filing fee	
114 150	214 75	Provisional filing fee	

SUBTOTAL (1) (\$)
690**2. EXTRA CLAIM FEES**

Total Claims	Extra Claims	Fee from below	Fee Paid
31	-20** = 11	78	198
6	-3** = 3	78	234
Multiple Dependent			

**or number previously paid, if greater; For Reissues, see below

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description
103 18	203 9	Claims in excess of 20
102 78	202 39	Independent claims in excess of 3
104 260	204 130	Multiple dependent claim, if not paid
109 78	209 39	** Reissue independent claims over original patent
110 18	210 9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) (\$)
432**FEE CALCULATION (continued)****3. ADDITIONAL FEES**

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
105 130	205 65	Surcharge - late filing fee or oath	
127 50	227 25	Surcharge - late provisional filing fee or cover sheet	
139 130	139 130	Non-English specification	
147 2,520	147 2,520	For filing a request for ex parte reexamination	
112 920*	112 920*	Requesting publication of SIR prior to Examiner action	
113 1,840*	113 1,840*	Requesting publication of SIR after Examiner action	
115 110	215 55	Extension for reply within first month	
116 380	216 190	Extension for reply within second month	
117 870	217 435	Extension for reply within third month	
118 1,360	218 680	Extension for reply within fourth month	
128 1,850	228 925	Extension for reply within fifth month	
119 300	219 150	Notice of Appeal	
120 300	220 150	Filing a brief in support of an appeal	
121 260	221 130	Request for oral hearing	
138 1,510	138 1,510	Petition to institute a public use proceeding	
140 110	240 55	Petition to revive - unavoidable	
141 1,210	241 605	Petition to revive - unintentional	
142 1,210	242 605	Utility issue fee (or reissue)	
143 430	243 215	Design issue fee	
144 580	244 290	Plant issue fee	
122 130	122 130	Petitions to the Commissioner	
123 50	123 50	Petitions related to provisional applications	
126 240	126 240	Submission of Information Disclosure Stmt	
581 40	581 40	Recording each patent assignment per property (times number of properties)	40
146 690	246 345	Filing a submission after final rejection (37 CFR § 1.129(a))	
149 690	249 345	For each additional invention to be examined (37 CFR § 1.129(b))	
179 690	279 345	Request for Continued Examination (RCE)	

Other fee (specify) _____

* Reduced by Basic Filing Fee Paid

SUBTOTAL (3) (\$)
40**SUBMITTED BY**

Name (Print/Type)	Lance R. Sadler
Signature	<i>Lance R. Sadler</i>

Registration No. (Attorney/Agent)
38,605**Complete (if applicable)**

Telephone	(502) 324-9256
Date	9/26/00

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Multi-Source Program Module Updater

Inventor:

Hanumant Yadav

ATTORNEY'S DOCKET NO. MS1-615US

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from U.S. Provisional Patent Application Serial No. 60/203450 entitled "Multi Source Copy Queue" filed on May 10, 2000.

TECHNICAL FIELD

This invention relates to the installation of program modules from more than one available source. In particular, this invention relates to the installation of updated program modules from one or more updated sources instead of from an older, standard source.

BACKGROUND

Described herein this document are issues and techniques related to the Microsoft® Windows® operating systems (such as Windows® 95, Windows® 98, Windows NT®, Windows® 2000, Windows® Me). However, those of ordinary skill in the art understand and appreciate that the issues and techniques described herein may be applicable to any operating system (OS) that support an expansive and updateable set of hardware devices. Examples of such operating systems include: OS/2™, Linux, Unix™, Mac OS™, and BeOS™.

In particular, the issues and techniques described herein are applicable to an OS that support plug-and-play (PnP) technology. PnP refers to the ability of a computer system (and its supporting OS) to automatically configure expansion boards and other hardware devices. With PnP, a user should be able to plug in a device and play with it, without worrying about setting DIP switches, jumpers, and other configuration elements.

Herein, an exemplary “OS” is discussed—this OS may be any OS having the issues discussed. The details of the implementation of a particular OS may differ from the exemplary OS described herein, but issues faced by such an implementation may be same or similar.

Initial Hardware-Specific Program Module Installation

Fig. 1 shows a typical personal computer 50. When first manufactured, a computer does not have an OS. Thus, the OS needs to be initially installed. In addition, the OS of existing computers are upgraded to a newer version of an OS.

Floppy disk(s) 62 and CD-ROM 64 represent potential OS installation source for computer 50. Also, server 30 represents a potential OS installation source for computer 50 over network 40.

Typically, an OS installation source (in a storage medium) contains a wide array of program modules—many of which are only installed if there is a need to do so. Block 66 illustrates examples of some of the program modules that might be found on an OS installation source: setup program 66a, core program modules 66b, and standard cabinets (“cabs”) 66c of hardware-specific program modules. Cabs are a series of linked compressed files in which hardware-specific program modules are stored. These hardware-specific modules are only necessary (and thus only need to be installed) when particular hardware is present in the computer.

This installation process is typically referred to as “setup.” The program that performs the installation is typically called the “setup program.” Program modules may be simply called “files” when referring to their status when stored on a storage medium.

During setup, program modules are unpacked and copied from the installation source onto the primary non-volatile storage system (such as a main hard drive 70 of Fig. 1) of the computer 50. This program-module duplication and organization has three major steps.

Core Files. First, the setup program copies the OS's core program modules (i.e., files) from the source represented by subblock 66b to a storage location 70a of the hard drive 70 of the computer 50. These files are the same core files installed into every computer regardless of the exact hardware configuration. An example of such a core file is the kernel of the OS.

Enumeration & Detection. Second, the setup program examines the hardware environment of the computer 50. It detects, enumerates, and identifies the hardware devices found during this step. It generates an enumerated list of program modules (such as a device drivers) that are associated the hardware devices found during this step. These program modules are located in the standard cabs 66c of the installation source 66.

Hardware-Specific Program Module Installation. Third, the setup program copies the program modules in the enumerated list. It copies them from the standard cabs 66c of the installation source 66 to a hardware-specific program module storage location 70b on the hard drive 70 of the computer 50.

Updating Installed Hardware-Specific Program Modules

Core and hardware-specific program modules are updated periodically to enhance features and functions and to fix known bugs and problems.

1 Occasionally, a collection of such updated program modules are released in a
2 package called a "service pack" (SP).

3 Typically, the service pack installation is very similar to the initial OS
4 installation except only those program modules that need updating are replaced by
5 updated modules in the SP. Because hardware-specific modules for other non-
6 installed devices do not exist on the computer 50, only the hardware-specific
7 modules of installed hardware devices are updated by the SP. After the SP
8 installation, the computer 50 has the latest core and hardware-specific program
9 modules for its current hardware configuration.

10 Some hardware-specific program modules are utilized by multiple
11 hardware devices. Therefore, updating one of those modules improves, corrects,
12 and/or enables the performance multiple devices.

13 SP may be delivered to the computer 50 in the same manner as the initial
14 installation. For example, it may be a CD-ROM 64, floppy disk 62, or over a
15 network 40 (from server 30).

16 **Later Hardware-Specific Program Module Installation**

17 Often hardware is added to computer 50 after the initial installation. These
18 hardware devices need program modules (such as device drivers) for the computer
19 and the OS to support them. Many of these hardware devices are Plug-and-Play
20 (PnP) and are automatically recognized and identified by the computer and its OS.
21 Each hardware device is associated with a hardware-specific program module
22 (such as device driver). Therefore, a PNP-capable computer and its OS attempt to
23
24
25

1 automatically install the hardware-specific program module associated with the
2 newly installed and PnP-recognized device.

3 Conventionally, the exemplary OS assumes that the original installation
4 source is the location from which to install the hardware-specific program
5 modules for the PnP-recognized device. Therefore, the hardware-specific modules
6 are drawn from the standard cabs 66c of the original source 66.

7 The new device installation process pulls the associated program modules
8 from the standard cabs 66c regardless of whether such modules are the most
9 current in light of the SP. The device installation does not provide an automatic
10 mechanism to check for the latest incarnation of a device's associated program
11 module. Although it may provide a user a manual choice for a source location,
12 this option is of little value unless the user knows from where to derive the latest
13 file.

14 Assuming that its associated modules are obtained from the original source,
15 a newly installed device on the computer 50 potentially has an old and possibly
16 functionally out-of-date program module. Furthermore, if this a common program
17 module used by multiple devices, then the function and operation of other devices
18 may fail or may result in incompatibilities.

19 20 **Conventional Approach**

21 Installation of an old hardware-specific program module when installing (or
22 reinstalling) a just-detected hardware device may result in a problem. The old
23 module has a potential to cause function failures, feature limitations, and/or
24 incompatibilities.
25

1 The conventional approach to solving this problem is to educate the user to
2 take additional steps after the just-installed device is installed. These additional
3 steps involve manually installing an updated version (perhaps from a service pack)
4 of the associated hardware-specific program module. This is a difficult approach
5 because users expect the computer to automatically install the correct associated
6 module because the installation is billed as plug-and-play (PnP).

8 SUMMARY

9 Described herein is a technology for automatically updating the most
10 current program modules associated with a just-detected hardware device.

11 In one described implementation, a program-module updater generates a list of to-
12 be-copied program modules. Typically, these modules are associated with just-
13 detected hardware devices. This implementation of the updater stores a data
14 structure for each module in such list. Each data structure includes an entry that
15 indicates the source location of the associated module. For example and typically,
16 the source location is the original source location for the installation of the
17 operating system.

18 The updater implementation examines the list to identify any of the listed
19 modules have been updated and it modifies the associated data structure of each
20 updated module so that a source entry in each data structure indicates the updated
21 source for the updated module. The updater copies all modules in the list to a
22 hardware-specific program module storage location of a computer. The source of
23 each module is indicated by its associated data structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like elements and features.

Fig. 1 is a block diagram of a computer/network environment with which an embodiment of a multi-source program module updater may be implemented.

Fig. 2 is a block diagram of an implementation of a multi-source program module updater.

Fig. 3 is a flow diagram showing a methodological implementation of a multi-source program module updater.

Fig. 4 is an example of a computing operating environment capable of implementing an implementation of a multi-source program module updater.

DETAILED DESCRIPTION

The following description sets forth specific embodiments of the multi-source program module updater that incorporate elements recited in the appended claims. These embodiments are described with specificity in order to meet statutory written description, enablement, and best-mode requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed present invention might also be embodied in other ways, in conjunction with other present or future technologies.

Incorporation by Reference

The following provisional application (from which priority is claimed) is incorporated by reference herein: U.S. Provisional Patent Application Serial No. . 60/203450 entitled "Multi Source Copy Queue" filed on May 10, 2000.

Introductory Terminology

Described herein are exemplary implementations of the program-module updater (i.e., "exemplary program-module updater"). A program module is a generic term for any a section of computer-executable instructions. A hardware-specific program module is a generic label for a program module that is associated with a specific hardware device and it is generally intended to facilitate computer interaction with such device. An example of such a hardware-specific program modules is a device driver.

General Exemplary Program-Module Updating Environment

Fig. 1 is described above (in the background section) as it is applicable to operating system (OS) installation and to conventional program-module updating. Now, Fig. 1 is described as it is applicable to the exemplary program-module updater.

Fig. 1 illustrates an overall program-module updating system 20. Such a system includes a subject computer 50 having conventional hardware components and an expandable, updateable OS. In this example, the computer 50 is a personal computer (PC) and the OS is a plug-and-play (PnP) OS, such as Microsoft® Windows® 98 SE.

1 The computer 50 is connected to a Program Module Update Server 30 via a
2 network 40. The update server 30 includes a content storage 32 and a distribution
3 server 34. The network 40 may be a local area network (LAN), wide-area network
4 (WAN), the Internet, or the like.

5 The computer 50 has multiple storage devices, such as floppy disk drive 52,
6 optical data drive 54, and a non-removable storage drive 70. A floppy disk 62
7 inserts into the floppy disk drive 52 of the computer. Likewise, an optical data
8 disk 64 inserts into the optical data drive 54 of the computer. The non-removable
9 storage drive 70 (e.g., hard drive) in the computer is not removable, as the name
10 suggests.

11 Using the exemplary program-module updater, update cabs 70d is copied
12 from the SP on the non-removable drive 70. Thus, the non-removable drive 70
13 includes core program modules 70a, hardware-specific program modules 70b,
14 other files 70c, update cabs 70d, and unused space 70e.

15 Assume that later a device is installed. This device has an updated
16 hardware-specific program module stored away in the update cabs 70d. Instead of
17 pulling an old hardware-specific program module from the standard cabs 66c, the
18 exemplary program-module updater will redirect the computer to access the
19 updated hardware-specific program module in the update cabs 70d on the hard
20 drive of the computer.

21 Alternatively, the update cabs may be stored in content storage 32 of the
22 Program Module Update Server (PMUS) 30. As it is needed, distribution server
23 34 sends the appropriate program module updates to the computer 50.
24
25

Exemplary Program-Module Updater

Fig. 2 shows an embodiment of the exemplary multi-source program-module updater 100. This embodiment may be implemented in software, hardware, or a combination thereof. An example of such software includes an OS, a setup application for an OS, an OS update application, and a software installation application.

This embodiment includes a list generator 110 for generating a list of to-be-copied program modules. This list is stored in list storage 112. Each entry in the list is an instance of a data structure with information regarding a program module. Included in that data structure is a source location (i.e., locus) from which the program module itself may be obtained.

A determination unit 120 examines the list of program modules and determines which modules have been updated since the original installation. It may do this by comparing the names of the modules in the to-be-copied list (stored in list storage 112) with another list. This other list includes the names of program modules that have been updated since the original installation. This other list may identify the updated source location for the updated modules.

A source-redirection unit 130 modifies the data structures of the updated program modules (identified by the determination unit). Such modification indicates the source location of the updated versions of the program modules.

A program-module copier 140 copies program modules from a source to a target. It copies program modules in the list stored in list storage 112. The target is defined. In Fig. 2, the target is Target disk 148. The source of each program module is determined by the source location indicated in the data structure of each

1 program module. For example, the source may be Source A disk 142, Source B
2 disk 144, or Source C disk 144.

3 4 **Methodological Implementation of Exemplary Program-Module Updater**

5 Fig. 3 shows a methodological implementation of the exemplary multi-
6 source program-module updater performed by the program-module updater 100.
7 This methodological implementation may be performed in software, hardware, or
8 a combination thereof.

9 At 210 of Fig. 3, the exemplary program-module updater generates a list of
10 to-be-copied program modules. Typically, this list of to-be-copied program
11 modules is generated when one or more hardware devices are installed on (or
12 initially detected by) the computer 50. Hardware-specific program modules are
13 associated with such new devices. Those modules are included in the list.

14 Assume an exemplary case where an OS has a program module called
15 “setupx.dll” implementing the exemplary program-module updater. Setupx.dll
16 called a main function called “VcpClose()” once an triggering event occurs—such
17 as an “inf”-based operation is performed (such an operation includes “copy files”,
18 “delete files”, “add registry”, “delete registry”). The VcpClose() function performs
19 the desired inf-based operation.

20 At 212 of Fig. 3, the exemplary updater stores a data structure for each
21 module in such list. Each data structure includes an entry that indicates the source
22 location of the associated module. For example and typically, the source location
23 is the original source location for the installation of the OS. It may be, for
24 example, standard cabs 66c of floppy disk 62.

1 In the “setupx.dll” example, each module in the list has an associated data
2 structure called LOGDISKDESC_S in the following format:

```
3  
4 typedef struct _LOGDISKDESC_S { /* ldd */  
5     WORD      cbSize;                // Size of this structure (bytes)  
6     LOGDISKID ldid;                // Logical Disk ID.  
7     LPSTR     pszPath;              // Ptr. to associated Path string.  
8     LPSTR     pszVolLabel;          // Ptr. to Volume Label string.  
9     LPSTR     pszDiskName;          // Ptr. to Disk Name string.  
10    WORD      wVolTime;              // Volume label modification time.  
11    WORD      wVolDate;              // Volume label modification date.  
12    DWORD      dwSerNum;             // Disk serial number.  
13    WORD      wFlags;                // Flags.  
14 } LOGDISKDESC_S, FAR *LPLOGDISKDESC;
```

15 Collectively, the “ldid”, “pszVolLabel”, and “pszDiskName” fields in the above
16 data structure indicate the source location for acquiring the program module
17 associated with a given instance of such structure.

18 At 214 of Fig. 3, the exemplary updater examines the list to identify any of
19 the listed modules have been updated (and thus their updated source is located in
20 the update cabs.) At 216, the exemplary updater modifies the associated data
21 structure of each updated module so that a source entry in each data structure
22 indicates the updated source for the updated module.

23 At 218 of Fig. 3, the exemplary updater copies all modules in the list to the
24 hardware-specific program module storage location 70b of the computer 50. The
25 source of each module is indicated by its associated data structure. Such source
26 may be the standard cabs 66c or the update cabs 70d.

27 In reference again to the “setupx.dll” example, when the VcpClose()
28 function performs a copy operation, it copies a specified list of program modules
29 (typically, device drivers and DLLs) from a source location to the hard drive of the
30 computer. Before any modules are copied, VcpClose() calls another function

called CheckAndPreserveFiles() to examine the list and determine if any of the listed modules have an updated source location.

A stored file may contain a list of all modules that have been updated. In addition, it may contain a list of all modules of the OS and their most current source. In this “setupx.dll” example, the OS contains a file called “layout.inf” that lists the names and locations of all program modules of the OS. In this example, it may be modified to include the locations of the updated source of each module.

If the CheckAndPreserveFiles() function finds such a module, it updates the “ldid”, “pszVolLabel”, and “pszDiskName” fields of the instance of the LOGDISKDESC_S data structure associated with such module. Once these change are made, the list is returned to the calling functions for normal processing.

Exemplary Alternative Implementations

Multiple Service Packs. Conventionally, the last applied SP overrides and overwrites any previous existing SPs. Hence, subsequent SPs must contain all updated program modules that were part of a previous SP. For example, if “kmixer.sys” was updated in service pack 1 (SP1), but service pack 2 (SP2) did not contain any changes for “kmixer.sys”, SP2 must include “kmixer.sys.” Otherwise, the updated version of “kmixer.sys” would be lost when SP2 was applied over SP1.

To ameliorate that problem, update cabs 70d may include multiple cabs from multiple SP updates. There may be multiple update cabs at different locations (e.g., over a network, on floppy, on hard disk, etc.) An implementation of a multi-source program module updater may support for multiple updates (e.g.,

multiple service packs). With each SP update, a new update cab is added or the existing update cab is updated.

With this alternative implementation, the registry may contain a list of modules and the name of the cab that contains the updated module. For example, the registry may contain the following entries:

Kmixer.sys, sp1.cab

Setupx.dll, sp2.cab

The exemplary updater may change the ldid, pszVolLabel and pszDiskName fields of the LOGDISKDESC_S structure based on the name of the program module and cab name combination when a match was found. As a result, the cab for SP 2 need not contain the SP 1 files. Thus, cabs may be smaller. Thus, download times may be quicker in case this SP is distributed over the Internet.

Multiple Lists. In another alternative implementation, an exemplary updater may maintain multiple separate lists. One list is for those modules from the standard cabs. Each of the others is from a particular update cab.

This implementation collectively copies all of the program modules from a given cab before moving on to the next cab. It does this until all cabs are copied.

Exemplary Computing Environment

Fig. 4 illustrates an example of a suitable computing environment 920 on which the exemplary program-module updater may be implemented.

1 Exemplary computing environment 920 is only one example of a suitable
2 computing environment and is not intended to suggest any limitation as to the
3 scope of use or functionality of the exemplary program-module updater. Neither
4 should the computing environment 920 be interpreted as having any dependency
5 or requirement relating to any one or combination of components illustrated in the
6 exemplary computing environment 920.

7 The exemplary program-module updater is operational with numerous other
8 general purpose or special purpose computing system environments or
9 configurations. Examples of well known computing systems, environments,
10 and/or configurations that may be suitable for use with the exemplary program-
11 module updater include, but are not limited to, personal computers, server
12 computers, thin clients, thick clients, hand-held or laptop devices, multiprocessor
13 systems, microprocessor-based systems, set top boxes, programmable consumer
14 electronics, network PCs, minicomputers, mainframe computers, distributed
15 computing environments that include any of the above systems or devices, and the
16 like.

17 The exemplary program-module updater may be described in the general
18 context of computer-executable instructions, such as program modules, being
19 executed by a computer. Generally, program modules include routines, programs,
20 objects, components, data structures, etc. that perform particular tasks or
21 implement particular abstract data types. The exemplary program-module updater
22 may also be practiced in distributed computing environments where tasks are
23 performed by remote processing devices that are linked through a communications
24 network. In a distributed computing environment, program modules may be
25

1 located in both local and remote computer storage media including memory
2 storage devices.

3 As shown in Fig. 4, the computing environment 920 includes a general-
4 purpose computing device in the form of a computer 930. The components of
5 computer 920 may include, by are not limited to, one or more processors or
6 processing units 932, a system memory 934, and a bus 936 that couples various
7 system components including the system memory 934 to the processor 932.

8 Bus 936 represents one or more of any of several types of bus structures,
9 including a memory bus or memory controller, a peripheral bus, an accelerated
10 graphics port, and a processor or local bus using any of a variety of bus
11 architectures. By way of example, and not limitation, such architectures include
12 Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA)
13 bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA)
14 local bus, and Peripheral Component Interconnects (PCI) bus also known as
15 Mezzanine bus.

16 Computer 930 typically includes a variety of computer readable media.
17 Such media may be any available media that is accessible by computer 930, and it
18 includes both volatile and non-volatile media, removable and non-removable
19 media.

20 In Fig. 4, the system memory includes computer readable media in the form
21 of volatile memory, such as random access memory (RAM) 940, and/or non-
22 volatile memory, such as read only memory (ROM) 938. A basic input/output
23 system (BIOS) 942, containing the basic routines that help to transfer information
24 between elements within computer 930, such as during start-up, is stored in ROM
25

1 938. RAM 940 typically contains data and/or program modules that are
2 immediately accessible to and/or presently be operated on by processor 932.

3 Computer 930 may further include other removable/non-removable,
4 volatile/non-volatile computer storage media. By way of example only, Fig. 4
5 illustrates a hard disk drive 944 for reading from and writing to a non-removable,
6 non-volatile magnetic media (not shown and typically called a "hard drive"), a
7 magnetic disk drive 946 for reading from and writing to a removable, non-volatile
8 magnetic disk 948 (e.g., a "floppy disk"), and an optical disk drive 950 for reading
9 from or writing to a removable, non-volatile optical disk 952 such as a CD-ROM,
10 DVD-ROM or other optical media. The hard disk drive 944, magnetic disk drive
11 946, and optical disk drive 950 are each connected to bus 936 by one or more
12 interfaces 954.

13 The drives and their associated computer-readable media provide
14 nonvolatile storage of computer readable instructions, data structures, program
15 modules, and other data for computer 930. Although the exemplary environment
16 described herein employs a hard disk, a removable magnetic disk 948 and a
17 removable optical disk 952, it should be appreciated by those skilled in the art that
18 other types of computer readable media which can store data that is accessible by a
19 computer, such as magnetic cassettes, flash memory cards, digital video disks,
20 random access memories (RAMs), read only memories (ROM), and the like, may
21 also be used in the exemplary operating environment.

22 A number of program modules may be stored on the hard disk, magnetic
23 disk 948, optical disk 952, ROM 938, or RAM 940, including, by way of example,
24
25

and not limitation, an operating system 958, one or more application programs 960, other program modules 962, and program data 964.

A user may enter commands and information into computer 930 through input devices such as keyboard 966 and pointing device 968 (such as a “mouse”). Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, serial port, scanner, or the like. These and other input devices are connected to the processing unit 932 through an user input interface 970 that is coupled to bus 936, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB).

A monitor 972 or other type of display device is also connected to bus 936 via an interface, such as a video adapter 974. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers, which may be connected through output peripheral interface 975.

Computer 930 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 982. Remote computer 982 may include many or all of the elements and features described herein relative to computer 930.

Logical connections shown in Fig. 4 are a local area network (LAN) 977 and a general wide area network (WAN) 979. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the computer 930 is connected to LAN 977 network interface or adapter 986. When used in a WAN

1 networking environment, the computer typically includes a modem 978 or other
2 means for establishing communications over the WAN 979. The modem 978,
3 which may be internal or external, may be connected to the system bus 936 via the
4 user input interface 970, or other appropriate mechanism.

5 Depicted in Fig. 4, is a specific implementation of a WAN via the Internet.
6 Computer 930 typically includes a modem 978 or other means for establishing
7 communications over the Internet 980. Modem 978, which may be internal or
8 external, is connected to bus 936 via interface 970.

9 In a networked environment, program modules depicted relative to the
10 personal computer 930, or portions thereof, may be stored in a remote memory
11 storage device. By way of example, and not limitation, Fig. 4 illustrates remote
12 application programs 989 as residing on a memory device of remote computer
13 982. It will be appreciated that the network connections shown and described are
14 exemplary and other means of establishing a communications link between the
15 computers may be used.

16 **Exemplary Operating Environment**

17 Fig. 4 illustrates an example of a suitable operating environment 920 in
18 which the exemplary program-module updater may be implemented. Specifically,
19 the exemplary program-module updater is implemented by any program 960-962
20 or operating system 958 in Fig. 4.

21 The operating environment is only an example of a suitable operating
22 environment and is not intended to suggest any limitation as to the scope of use of
23 functionality of the exemplary program-module updater described herein. Other
24
25

1 well known computing systems, environments, and/or configurations that may be
2 suitable for use with the exemplary program-module updater include, but are not
3 limited to, personal computers, server computers, hand-held or laptop devices,
4 multiprocessor systems, microprocessor-based systems, programmable consumer
5 electronics, wireless communications equipment, network PCs, minicomputers,
6 mainframe computers, distributed computing environments that include any of the
7 above systems or devices, and the like.

8 9 **Computer-Executable Instructions**

10 An implementation of the exemplary program-module updater may be
11 described in the general context of computer-executable instructions, such as
12 program modules, executed by one or more computers or other devices.
13 Generally, program modules include routines, programs, objects, components, data
14 structures, etc. that perform particular tasks or implement particular abstract data
15 types. Typically, the functionality of the program modules may be combined or
16 distributed as desired in various embodiments.

17 18 **Computer Readable Media**

19 An implementation of the exemplary program-module updater may be
20 stored on or transmitted across some form of computer readable media. Computer
21 readable media can be any available media that can be accessed by a computer.
22 By way of example, and not limitation, computer readable media may comprise
23 computer storage media and communications media.

Computer storage media include volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computer.

Communication media typically embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal such as carrier wave or other transport mechanism and included any information delivery media. The term “modulated data signal” means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared, and other wireless media. Combinations of any of the above are also included within the scope of computer readable media.

Conclusion

Although the multi-source program-module updater has been described in language specific to structural features and/or methodological steps, it is to be understood that the improved program-module updater defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed present invention.

1 **CLAIMS:**

2 1. A program-module update system, comprising:
3 a determination unit for determining whether a hardware-specific program
4 module is an updated program module;
5 a source-redirection unit for specifying a source locus for a program
6 module determined to be an updated program module by the determination unit.

7
8 2. A system as recited in claim 1 further comprising a list generator for
9 providing a list of hardware-specific program modules, wherein the determination
10 unit determines whether a module listed in such list is an updated module.

11
12 3. A system as recited in claim 1 further comprising a program-module
13 copier for copying a hardware-specific program module from the specified source
14 locus to a target locus.

15
16 4. A system as recited in claim 1, wherein the source locus is on a non-
17 removable storage medium.

18
19 5. A system as recited in claim 1, wherein the source locus is on a
20 removable storage medium.

1 6. A system as recited in claim 1, wherein the source locus is on a
2 storage medium remotely connected to the program-module update system via a
3 network.
4

5 7. A software installation application comprising a program-module
6 update system as recited in claim 1.
7

8 8. An operating system update application comprising a program-
9 module update system as recited in claim 1.
10

11 9. An operating system comprising a program-module update system
12 as recited in claim 1.
13

14 10. A program-module update system, comprising:
15 a source-redirection unit for specifying a source locus for a hardware-
16 specific program module to be copied to a target locus;
17 a program-module copier for copying the program module from the
18 specified source locus to the target locus.
19

20 11. A system as recited in claim 10 further comprising a determination
21 unit for determining whether a hardware-specific program module is an updated
22 program module so that the source-redirection unit specifies a locus for modules
23 determined to be an updated module by the determination unit.
24
25

1 12. A system as recited in claim 10, wherein the source locus is on a
2 non-removable storage medium.

3
4 13. A system as recited in claim 10, wherein the source locus is on a
5 removable storage medium.

6
7 14. A system as recited in claim 10, wherein the source locus is on a
8 storage medium remotely connected to the program-module update system via a
9 network.

10
11 15. A software installation application comprising a program-module
12 update system as recited in claim 10.

13
14 16. An operating system comprising a program-module update system
15 as recited in claim 10.

16
17 17. A method of updating a program module, the method comprising:
18 determining whether a hardware-specific program module is an updated
19 program module;
20 specifying a source locus for a program module determined to be an
21 updated program module by the determining.

1 **18.** A method as recited in claim 17 further comprising:
2 generating a list of hardware-specific program modules;
3 providing such list to the determining.
4

5 **19.** A method as recited in claim 17 further comprising copying a
6 hardware-specific program module from the source locus specified by the
7 specifying to a target locus.
8

9 **20.** A method as recited in claim 17, wherein the source locus is on a
10 non-removable storage medium.
11

12 **21.** A method as recited in claim 17, wherein the source locus is on a
13 removable storage medium.
14

15 **22.** A method as recited in claim 17, wherein the source locus is on a
16 storage medium remotely connected via a network.
17

18 **23.** A computer-readable medium having computer-executable
19 instructions that, when executed by a computer, performs the method as recited in
20 claim 17.
21
22
23
24
25

1 **24.** A computer-readable medium having computer-executable
2 instructions that, when executed by a computer, perform a method of updating
3 program modules, the method comprising:

4 determining whether a hardware-specific program module is an updated
5 program module; and

6 specifying a source locus for a program module determined to be an
7 updated program module by the determining.

8
9 **25.** A modulated signal updating a program module, the modulated
10 signal generated in accordance with the following acts:

11 determining whether a hardware-specific program module is an updated
12 program module; and

13 specifying a source locus for a program module determined to be an
14 updated program module by the determining.

1 **26.** A method of updating a program module, comprising:

2 obtaining a list of program-module data structures, each data structure
3 being associated with a hardware-specific program module and identifying a
4 source locus where the associated module is stored;

5 examining such list;

6 determining whether a program module associated with a data structure is
7 an updated program module; and

8 modifying the data structure associated with a program module determined
9 to be an updated program module by the determining so that a new source locus is
10 identified in the associated data structure.

11
12 **27.** A method as recited in claim 26 further comprising copying a
13 hardware-specific program module from the source locus identified in the data
14 structure associated with the program module to a target locus.

15
16 **28.** A method as recited in claim 26, wherein the source locus identified
17 in a data structure associated with a program module is on a non-removable
18 storage medium.

19
20 **29.** A method as recited in claim 26, wherein the source locus identified
21 in a data structure associated with a program module is on a removable storage
22 medium.

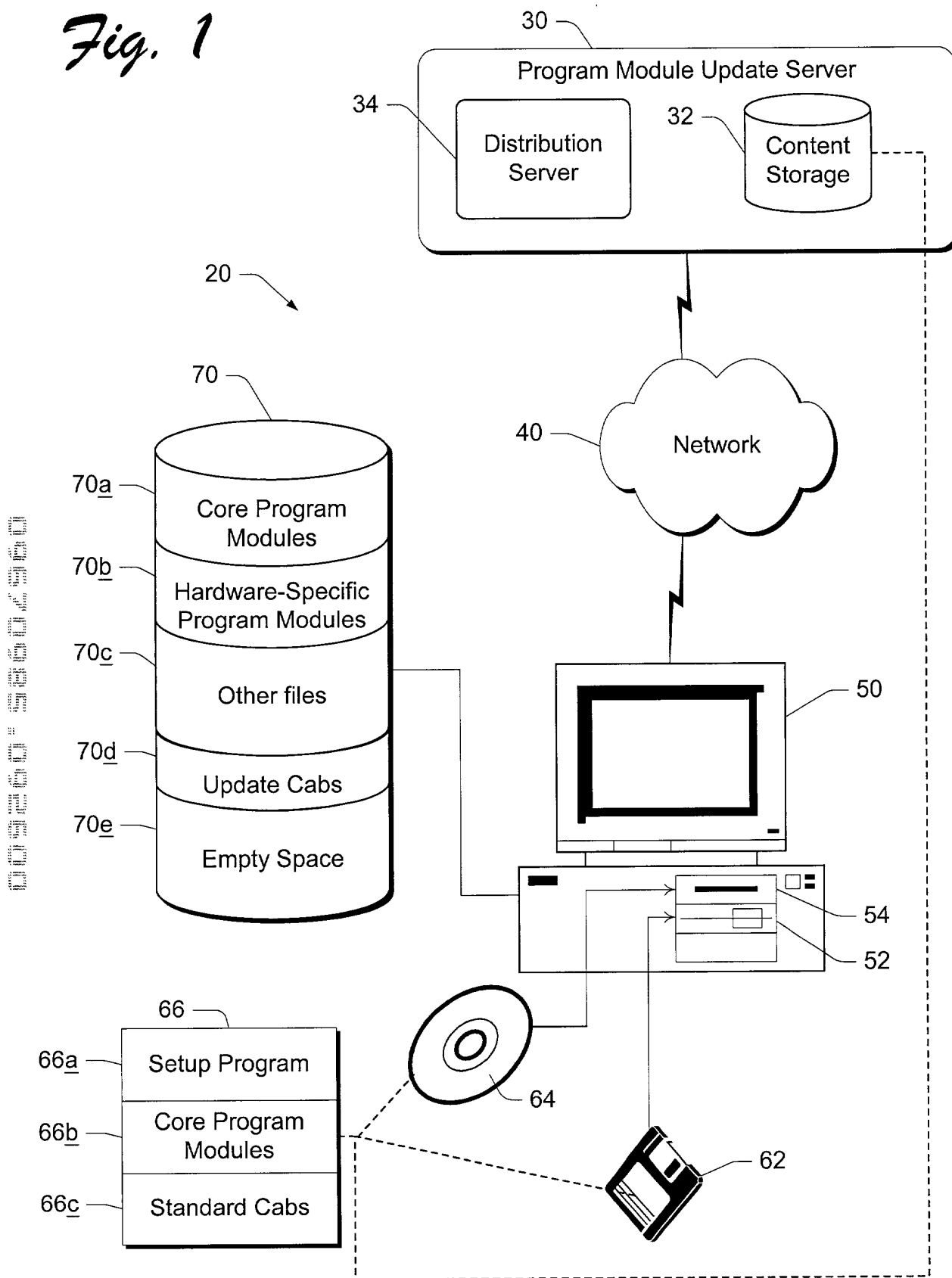
1 **30.** A method as recited in claim 26, wherein the source locus identified
2 in a data structure associated with a program module is on a storage medium
3 remotely connected via a network.
4

5 **31.** A computer-readable medium having computer-executable
6 instructions that, when executed by a computer, performs the method as recited in
7 claim 26.
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

1 **ABSTRACT**

2 Described herein is a technology for automatically updating the most
3 current program modules associated with a just-detected hardware device. In one
4 described implementation, a program-module updater generates a list of to-be-
5 copied program modules. Typically, these modules are associated with just-
6 detected hardware devices. This implementation of the updater stores a data
7 structure for each module in such list. Each data structure includes an entry that
8 indicates the source location of the associated module. For example and typically,
9 the source location is the original source location for the installation of the
10 operating system. The updater implementation examines the list to identify any of
11 the listed modules have been updated and it modifies the associated data structure
12 of each updated module so that a source entry in each data structure indicates the
13 updated source for the updated module. The updater copies all modules in the list
14 to a hardware-specific program module storage location of a computer. The
15 source of each module is indicated by its associated data structure.

16
17
18
19
20
21
22
23
24
25

Fig. 1

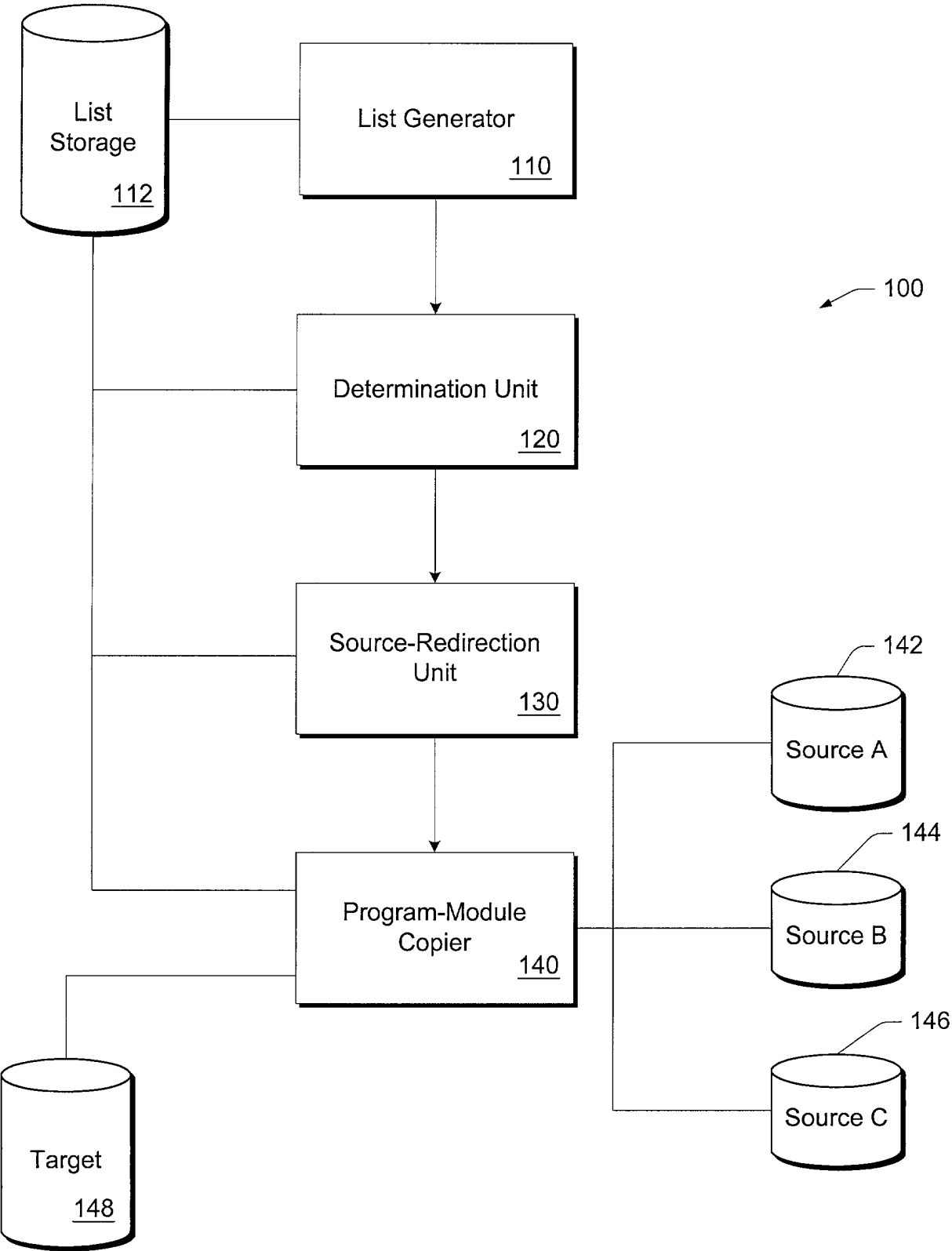
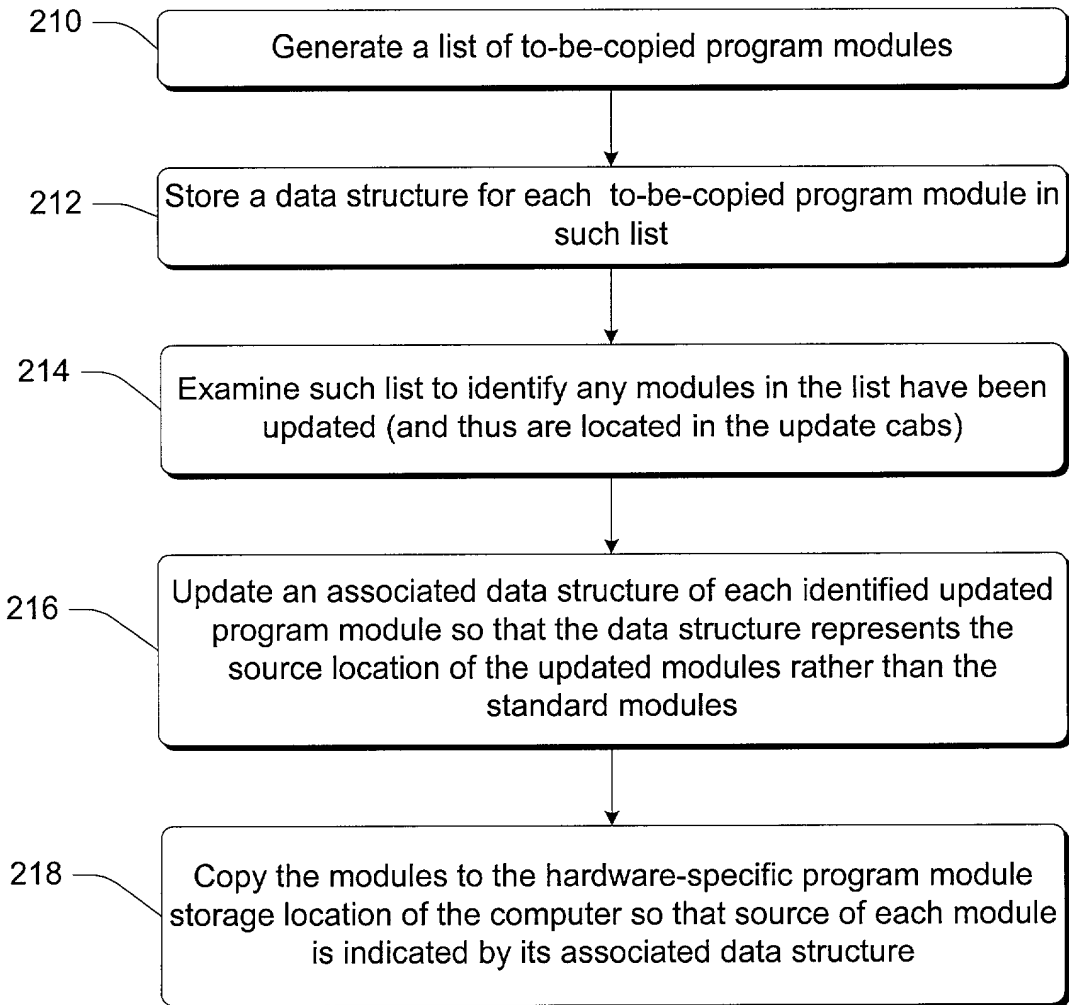
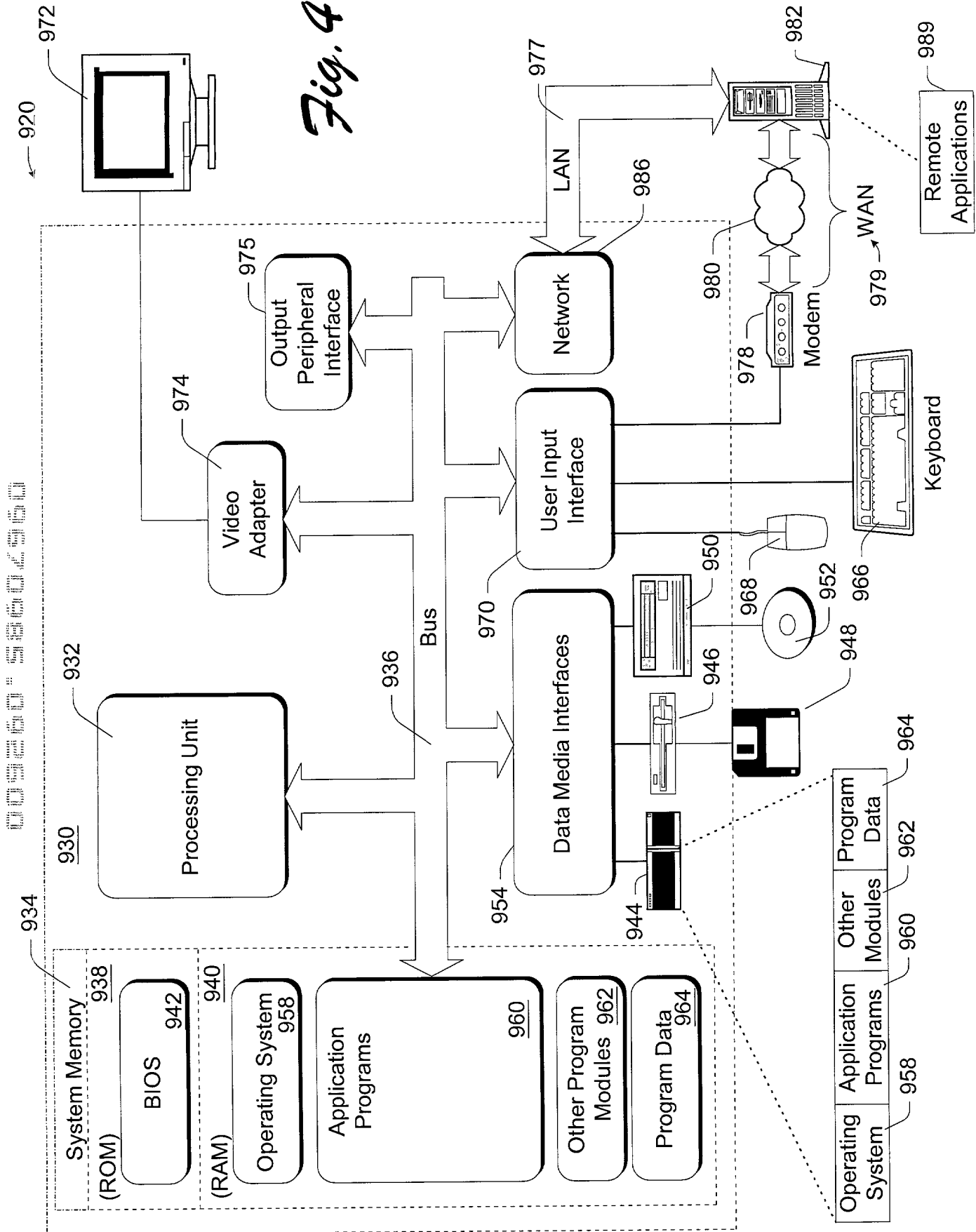


Fig. 2

*Fig. 3*



1 **IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

2 Inventorship..... Yadav
3 Applicant Microsoft Corporation
4 Attorney's Docket No. MS1-615US
5 Title: Multi-Source Program Module Updater

6 **DECLARATION FOR PATENT APPLICATION**

7 As a below named inventor, I hereby declare that:

8 My residence, post office address and citizenship are as stated below next to
9 my name.

10 I believe I am the original, first and sole inventor (if only one name is listed
11 below) or an original, first and joint inventor (if plural names are listed below) of the
12 subject matter which is claimed and for which a patent is sought on the invention
13 entitled "Multi-source Program Module Updater," the specification of which is
14 attached hereto.

15 I have reviewed and understand the content of the above-identified
16 specification, including the claims.

17 I hereby claim benefit under 35 U.S.C. 119(e) of United States Provisional
18 Application 60/203,450, filed May 10, 2000.

19 I acknowledge the duty to disclose information which is material to the
20 examination of this application in accordance with Title 37, Code of Federal
21 Regulations, § 1.56(a).

22 PRIOR FOREIGN APPLICATIONS: no applications for foreign patents or
23 inventor's certificates have been filed prior to the date of execution of this
24 declaration.
25

Power of Attorney

I appoint the following attorneys to prosecute this application and transact all future business in the Patent and Trademark Office connected with this application: Lewis C. Lee, Reg. No. 34,656; Daniel L. Hayes, Reg. No. 34,618; Allan T. Sponseller, Reg. 38,318; Steven R. Sponseller, Reg. No. 39,384; James R. Banowsky, Reg. No. 37,773; Lance R. Sadler, Reg. No. 38,605; Michael A. Proksch, Reg. No. 43,021; Thomas A. Jolly, Reg. No. 39,241; David A. Morasch, Reg. No. 42,905; Kasey C. Christie, Reg. No. 40,559; Nathan R. Rieth, Reg. No. 44,302; Brian G. Hart, Reg. No. 44,421; Katie E. Sako, Reg. No. 32,628 and Daniel D. Crouse, Reg. No. 32,022.

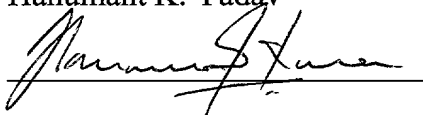
Send correspondence to: LEE & HAYES, PLLC, 421 W. Riverside Avenue, Suite 500, Spokane, Washington, 99201. Direct telephone calls to: Kasey C. Christie (509) 324-9256.

All statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statement may jeopardize the validity of the application or any patent issued therefrom.

Full name of inventor:

Hanumant K. Yadav

Inventor's Signature



Date: 09/21/2000

Residence:

Bellevue, WA

Citizenship:

US

Post Office Address:

16434 SE 9th Street
Bellevue, WA 98008